Experiment #2: Digital Input Signal Conditioning

Process control relies on gathering input information, evaluating it, and initiating action. In industrial control, input information most often involves monitoring field devices whose outputs are one of two possible states. A switch is the most common example of a "bi-state" device. It is either open or closed.

Switches can provide control of an operation in three ways. One may be wired directly with the load and therefore control the full current and voltage. A switch also can be wired in the input circuit of a relay. In this case, the switch controls the relay's relatively low power input and the output contacts control load power. The on/off status of a switch may also provide a digital input to a programmable controller.

How many switches have you used today? And, what processes were affected by the toggling of those switches? Table 2.1 lists a few possibilities, starting at the beginning of your day:

Table 2.1: Switch Possibilities at the Beginning of your Day

Switch Status	Result
First, you may slap the "SNOOZE" button on your alarm clock.	The buzzing stops and -- Ah! 5 more minutes of sleep!
Next, stumble to the bathroom and flip "ON" the bathroom light.	Ouch! Turn it "OFF." Those vanity lights hurt!
Now, into the kitchen, start your coffeemaker, press down the toaster, and program your microwave. Open the refrigerator and the light comes on .	Breakfast is ready. And who knows if that light really goes off when you close the refrigerator?
Turn on the thermostat.	Heat or AC – your choice. What temperature? A setpoint is usually just a "switching point."
Turn on your TV, change the channel, turn up the volume.	The pushbuttons on the front or the flashing infrared LED in your remote– they all still just switch data.
Make a phone call. Lift the receiver and check for dial tone. Key in the phone number.	The limit switch held down by the handset now is in its "off the hook" position. Each switch on the keypad allows a specific tone to be generated.
Boot your PC. Switch on the monitor. Left click the mouse to check your e-mail.	These are only three obvious ones. There are many more switches behind the scenes in your PC.
You are up to 15 switches and you haven't even left your house!	

Some of the switches listed in Table 2.1 probably have direct control of electrical continuity to the loads involved. For example, the bathroom light switch controls the actual current flowing to the vanity light bulbs. The thermostat is an example of a switch controlling a low-voltage system that controls a relay in your furnace or air conditioner.

Most of the switches in Table 2.1, however, probably are providing a digital high or low signal being monitored by an electronic control system. It is the status of this input signal that is evaluated and used to determine the appropriate state of the outputs involved. The snooze button isn't physically opening the alarm circuit of your clock radio. When you "slapped" it, the momentary change of state was recognized by a programmable circuit. As a result, the program instructed the output to go off and add five minutes to the programmed alarm time. The start button on your microwave doesn't have to carry the actual current that powers the magnatron, inside light, and ventilation fan. However, pressing it creates an input causing the oven's microcontroller to close relays that do handle these loads.

Most often we think of switches as mechanical devices that make and break continuity between contact points in a circuit. In the case of the manual pushbutton and the limit switches pictured in Figure 2.1, this is exactly the case.

Figure 2.1: A Variety of Manual Pushbutton and Mechanical Limit Switches

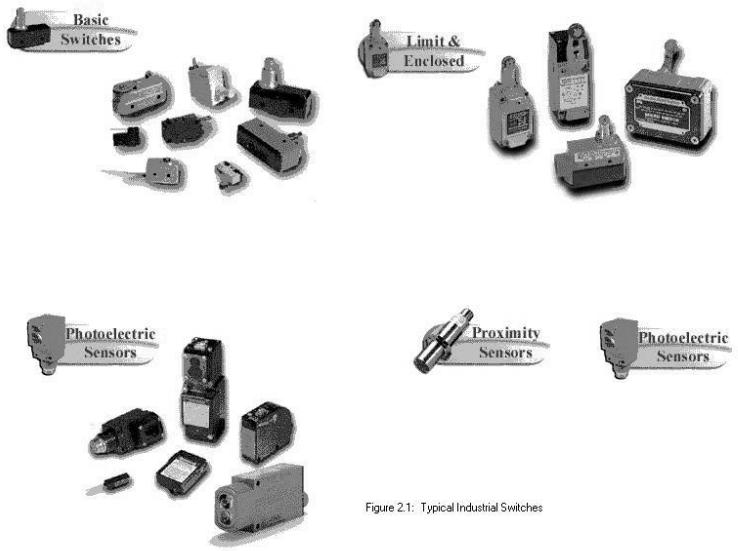
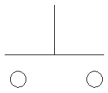
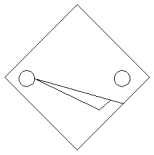
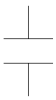
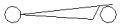
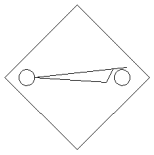


Figure 2.1: Typical Industrial Switches

Table 2.2 shows the schematic representation of various industrial switches. The symbols are drawn to represent the switch's "normal" state. Normal state refers to the unactuated or rest state of the switch. The pushbutton switches in this exercise kit are Normally Open (N.O.). Pressing the pushbutton results in a plunger shorting the contacts. The resistance goes from its open value of nearly infinite ohms to a value very near zero. A similar mechanism produces a like action in a Normally Open limit switch.

Table 2.2: Schematic Representation of Various Industrial Switches

	Push-button	Mechanical Limit	Proximity Switch	Relay Contacts
Normally Open				
Normally Closed				

While the concept of the switch doesn't get any simpler, there seems to be no limit to the physical design of switches that you will find in industrial control applications. Switches also may be designed as Normally Closed (N.C.); they are closed when at rest and actuation causes their contacts to open. As a technician, programmer, or system designer, you must be aware of the Normal (resting) position of a switch.

Figure 2.2: Schematic Representation of Pushbutton Switches

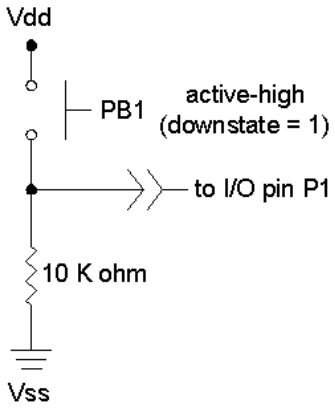


Figure 2.2a

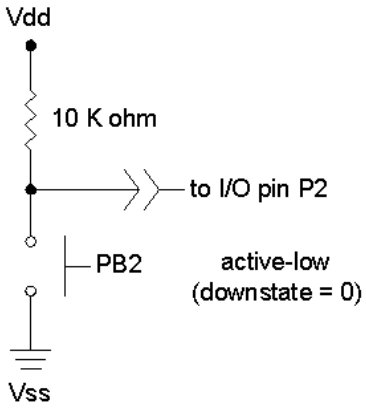


Figure 2.2b

What's a...

Digital Input (TTL, CMOS, ECL, etc.)?

Logic devices are built with a variety of processes that operate at different voltages. The manufacturer's datasheet will list several critical values for each device. Absolute Maximum Ratings are voltages and currents which must not be exceeded to avoid damaging or destroying the chip. I/O pins on the BASIC Stamp II should not exceed 0.6 V or Vdd+0.6 V (5.6V) with respect to Vss.

The logic transition between high and low is specified in the DC characteristics of the datasheet. A voltage of 0.2 Vdd (1 V on the BASIC Stamp II) is guaranteed to be low, which 0.45 Vdd (2.25 V) or higher is guaranteed to be high. There is a gray area between these two voltages where the actual transition will occur. It is dependant on temperature and supply voltages where the actual transition will occur. It also varies with temperature and supply voltage but will normally occur at about 1.4 volts.

The input pins of the BASIC Stamp do not detect "changes in resistance" between the switch's contacts. These inputs expect appropriate voltage levels to represent a logic high or a logic low. Ideally, these levels would be +5 volts for a logic high (1) and 0 volts for a logic low (0).

To convert the two resistive states of the switch into acceptable inputs, it must be placed in series with a resistor across the +5 volt supply of the BASIC Stamp. This forms a voltage divider circuit in which the resistive status of the switch is compared to the resistive value of the reference resistor. Figure 2.2 shows the two possibilities for our simple N.O. pushbutton switch. Figure 2.2a will result in +5 volts being fed to the input pin when it is pressed. When the switch is open, there is no continuity; therefore, no current flows through the 10K resistor and the input pin is grounded.

What's a...

Reference Resistor:

The 10K-ohm fixed resistor in Figures 2.2a and 2.2b is required to get dependable logic levels. It is wired in series with the switch. Its value must be much greater than the closed resistance of the switch and much less than its open resistance. When the switch is open in Figure 2.2a, the resistor gets no voltage and the input point is "pulled down" to ground. In Figure 2.2b, the open switch causes the input to be "pulled up" to +5 volts. You must consider the use of pull-up and pull-down resistors when working with all mechanical switches and some electronic switches.

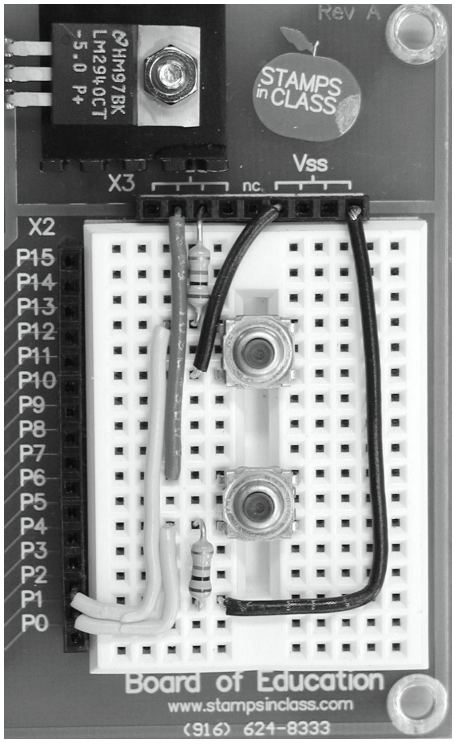
In Figure 2.2b, the switch closure results in grounding of the input pin. Zero volts is a logic low. When the switch is opened, there is again no voltage drop across the 10K-ohm resistor and the voltage at the input is +5, a logic high. The circuits are essentially the same, although the results of pressing the switch are exactly opposite. From a programming standpoint, it is important to know with which configuration you are dealing.

Exercises

Exercise #1: Switch Basics

To begin an investigation of programming for simple switch activity, wire the two pushbutton switches shown in Figure 2.2 onto the Board of Education breadboard. Connect the active-high configuration (Figure 2.2a) to I/O Pin 1 and the output of the active-low configuration (Figure 2.2b) to Pin 2. Note which one is which. As stated earlier, this is important. Figure 2.3 shows a pictorial of how the circuit is built on the Board of Education.

Figure 2.3: Pictorial of Parts Layout for circuits of Figure 2.2

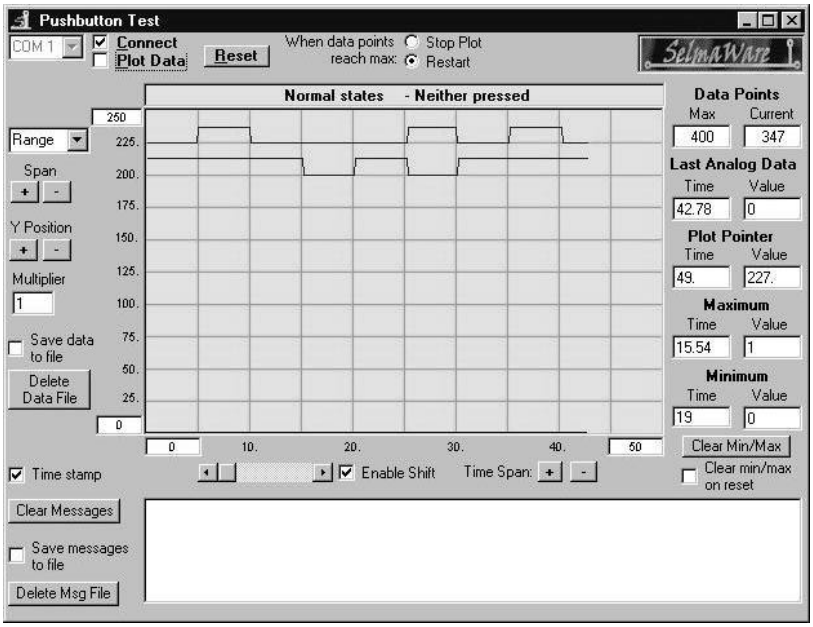


The following program is written to use the StampPlot Lite interface for displaying the status of the switches. The procedure will be the same as you followed in Experiment #1, Flowcharting and StampPlot Lite. First, enter Program 2.1. You may omit from the program all comments which include the apostrophe (') and the text that follows.

```
'Program. 2.1:  Switch Level Detection with StampPlot Lite Interface

debug "!TITL Pushbutton Test",13        ' Titles the StampPlot screen

input 1                                 ' Set P1 as an input
input 2                                 ' Set P2 as an input

Loop:
  pause 10                              ' Slow the program loop
  debug ibin in1, bin in2, 13           ' Plot the digital status
  debug dec 0, 13                       ' Output a 0 to allow for screen shift
  if (in1 = 1) and (in2 = 0) then both  ' Test for both pressed
  if in1 = 1 then PB1                   ' Test if active-high PB1 is pressed
  if in2 = 0 then PB2                   ' Test if active-low PB2 is pressed
debug "!USRS Normal states    - Neither pressed", 13
                                        ' Report none pressed
goto Loop

PB1:                                    ' Report PB1 pressed
  debug "!USRS Input 1 is High  - PB 1 is pressed ", 13
goto Loop

PB2:                                    ' Report PB2 pressed
  debug "!USRS Input 2 is Low   - PB 2 is pressed ", 13
goto Loop

Both:                                   ' Report both pressed
  debug "!USRS P1 High & P2 Low - Both pressed",  13
  debug "!BELL", 13                     ' Sound the bell.
goto Loop
```

Run the program. **Debug** will scroll the switch status and the input's digital value. Close the debug screen and open StampPlot Lite. Select the appropriate COM port and check the Connect and Plot Data boxes. Press the reset switch on your Board of Education and the trace of **in1** and **in2** should start across the screen. Your display should look similar to Figure 2.4. Press the pushbuttons and become familiar with the operation of your system. Next, we will look at how the program works.

Figure 2.4: Typical Screen Shot of StampPlot Monitoring the Status of Pushbuttons



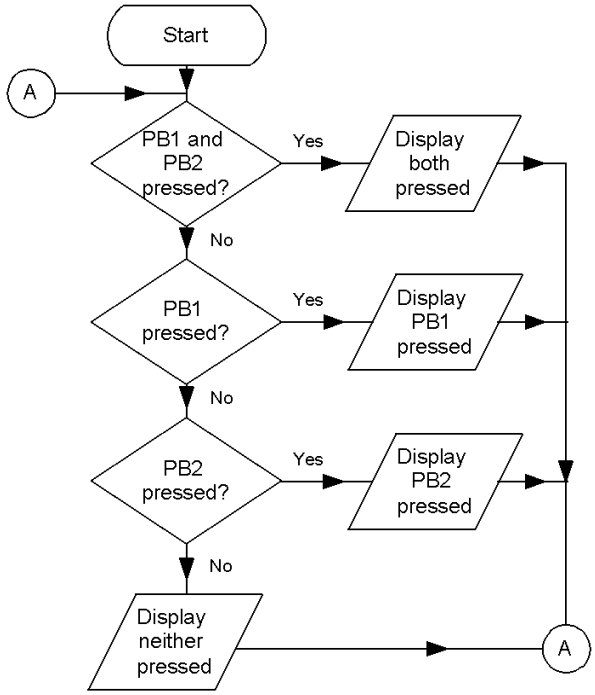
The purpose of this program is to run code based on the pressed or not-pressed condition of the two pushbuttons. This simple exercise gives insight to several considerations when dealing with digital inputs, programming multiple if-then statements, and using some of the PBASIC logical operators.

First, the statements in1 and in2 simply return the logic value of the input pins: +5 V = logic 1 and 0 V = logic 0. The active-high PB1 returns a 1 if pressed. The active-low PB2 returns a 0 when it is pressed. The program is testing for the "logical" status of the inputs; as the programmer, you must understand how this correlates to the "pressed" or "not pressed" condition of the pushbuttons involved. This is evident in the first line of the program loop where the logic operator AND is being used.

When you consider our switch configurations, it makes logical sense that if **in1** returns a logic high and **in2** returns a logic low then both switches are pressed. Output actions of industrial controllers often are dependent upon the status of multiple switches and contacts. A review of the PBASIC logical operators, including **and**, **or**, **xor** and **not,** can provide useful tools in meeting these requirements using the BASIC Stamp.

Another aspect of Program 2.1 is to notice the flow of the program loop. The **if-then** structure tests for a condition and **if** the condition is met, **then** the program execution is passed to the **label**. In this case, the label routine simply prints the conditions of the switches to the StampPlot Lite Status box. In industrial applications, this portion of the program would cause the appropriate output action to occur. Since the last line of each label is **goto loop**, program execution returns to the top of the loop and any code below that **if-then** statement is circumvented. The flowchart in Figure 2.5 shows how the program executes.

Figure 2.5: Flowchart for Program 2.1



If both switches are pressed, "**`if (in1 =1) and (in2 = 0`**)" is true. Program execution then would go to the **`both`** label. The "both pressed" condition would be indicated in the User Status Bar and your computer bell would ring. After this, program execution is instructed to go back to **`loop`** and test the switches again. As long as both switches remain pressed, the result of this test is continually true and looping is occuring only within this part of the program.

If either or both switches become not pressed, the next three lines of code will do a similar test for the condition.

Pressing PB1 results in "**`if in1 = 1`**" being true, execution is passed to the PB1 label action, and a return to the top of the loop; "**`if in2 = 0`**" is never tested. Is this good or bad? Neither, really. But, understanding the operation of multiple **`if-then`** statements can be a powerful tool for programming applications. Forgetting this can result in frustrating and not-so-obvious bugs in your program. For instance, what would happen in our program if the test for both switches being pressed "**`if (in1 =1) and (in2 = 0) then both"`** was put after the individual switch tests?

Quick Challenge

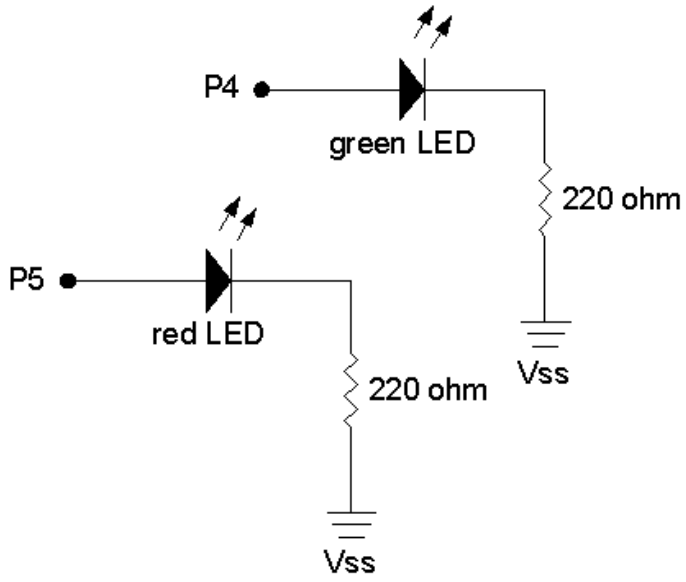
While running the program, try to reproduce the switch status shown in the screen shot of Figure 2.4.

Exercise #2 – Switch Bounce and Debouncing Routines

In the previous exercise, the steady-state level of the switch was being reported. The routine of reporting the switch status was performed on each program loop. What if you wanted to quickly press the switch and have something occur only once? There are two issues with which to contend. The first is: How quickly can you press and release the switch? You have to do it within the period of one program cycle. The second problem is contending with switch bounce. Switch bounce is the tendency of a switch to make several rapid on/off actions at the instant it is pressed or released.

The following program will demonstrate the difficulty in accomplishing this task. Two light-emitting diodes have been added as output indicators on Pin 4 and Pin 5. Wire the LEDs relative to Figure 2.6.

Figure 2.6: Active-High LED Circuit to be Added to the Schematic in Exercise #1



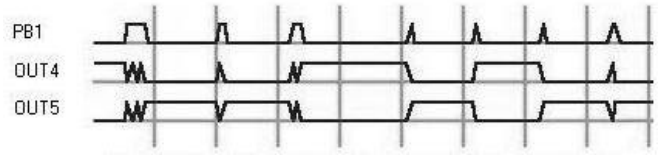
Enter and run the program according to StampPlot Lite procedures. The status of the pushbutton and the LEDs is being indicated. When PB1 is pressed, the LEDs will toggle. Can you be quick enough to make them toggle only once on alternate presses? Try it.

```
'Program 2.2 No Debouncing

pause 500
debug "!TITL Toggle Challenge",13' Titles the StampPlot screen
debug "!TMAX 25", 13                    ' Sets the plot time (seconds)
debug "!PNTS 300", 13                   ' Sets the number of data points

input 1                                 ' Set P1 as an input
input 2                                 ' Set P2 as an input
output 4                                ' Red LED
out4 = 1                                ' Initialize ON
output 5                                ' Green LED
out5 = 0                                ' Initialize OFF

Loop:
  debug ibin in1, bin in4, bin in5, 13  ' Plot the digital status.
  debug dec 0, 13                       ' Output a 0 to allow for screen shift
  if in1 = 1 then Action                ' Test the switch
                                        ' Optional pause 5 if StampPlot locks up
goto Loop

Action:                                 ' Toggle last state
  toggle 4
  toggle 5
GOTO Loop
```

If StampPlot Lite isn't responding to data sent by the BASIC Stamp, you may need to insert a very short delay in the **loop:** routine. A **pause 2** or **pause 5** (even up to 10 on slower computers) will alleviate any transmission speed problems you may encounter.

It is nearly impossible to press and release the pushbutton fast enough to perform the action only once. The problem is twofold as Figure 2.7 indicates. The program loop executes very fast. If you are slow, the program has a chance to run several times while the switch is closed. Add to this several milliseconds of switch bounce, and you may end up with several toggles during one press.

Figure 2.7: Slow Response and no Debounce Can be a Problem

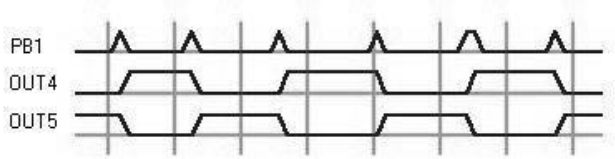


Further slowing the execution time of the program loop can help remedy the problem. (If the above program didn't work properly with StampPlot Lite, a delay in execution speed will allow for serial data transmission). Add a delay of 250 milliseconds to the **`Action:`** routine. This allows 250 milliseconds for the switch to settle after closing and then return to its open position.

Modify your program to include "Pause 250" to increase the loop time and negate switch bounce.

```
Program 2.3 (modify program 2-2 to slow it down)
Action:                    ' Toggle last state
  toggle 4
  toggle 5
  pause 250                ' Added to allow for settling time

goto loop
```

Figure 2.8: Adding a Pause Makes the Toggle Challenge Much Easier



By allowing settling time and pressing the button quickly, you make it much easier to get the **`Action:`** to occur only once. This technique helps debounce the switch and gives you enough time to release it before the next program cycle. The **`pause`** must be long enough to allow for these factors. If the **`pause`** is too long, however, a switch closure may occur and never be seen.

Exercise #3 – Edge Triggering

Counting routines pose additional problems for digital input programming. Exercise #2 used the **pause** command to eliminate switch bounce, which is compounded in industrial applications such as counting products on a conveyor. Not only does the switch have inherent bounce, but the product itself may have irregular shape, be wobbling, or stop for some time while activating the switch. There may be only one product, but the switch may open and close several times. Also, if the one product stays in contact with the switch for several program loop cycles, the program still should register it only once, not continually like in Program 2.2.

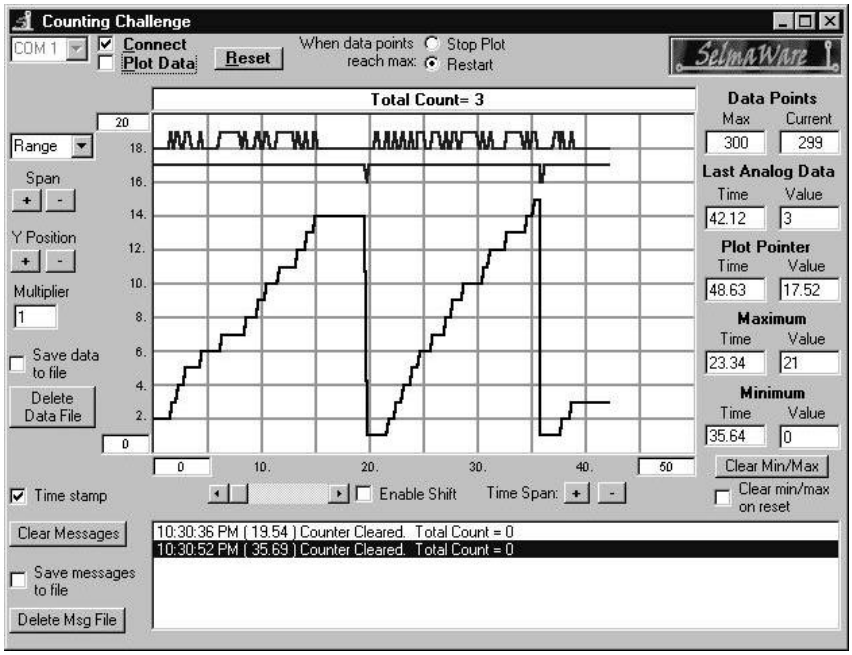
Program 2.4 uses a flag variable to create a program that responds to the initial low-to-high transitions of the switch. Once this "leading edge" of the digital input is detected, **Action:** will be executed. Then the flag will be set to prevent subsequent executions until the product has cleared and the switch goes low again. Enter Program 2.4.

```
' Program 2.4: Switch Edge Detection
' Count and display the number of closures of PB1.
' Reset total count with a closure of PB2.

pause 500
debug "!TITL Counting Challenge",13     ' Titles the StampPlot screen
debug "!TMAX 50",13                     ' Sets the plot time (seconds)
debug "!PNTS 300",13                    ' Sets the number of data points
debug "!AMAX 20",13                     ' Sets vertical axis (counts)
debug "!MAXR",13                        ' Reset after reaching maximum data points

input 1
input 2
Flag1 var bit                           ' flag for PB1
Flag2 var bit                           ' flag for PB2

Counts var word                         ' word variable to hold count
Flag1= 0                                ' clear the flags and Counts
Flag2 = 0
Counts = 0

Loop:
  pause 50
  debug "!USRS Total Count = ",dec Counts,13
  ' Display total counts in Status box
  debug dec Counts, 13                  ' Show counts on analog trace
  debug ibin in1, bin in2,13                  ' Plot the digital status.
```

```
  if in1 = 1 then Count_it              ' If pressed, count and display
  Flag1 = 0                             ' If not pressed, reset flag to 0
  if in2 = 0 then Clear_it              ' If PB2 is pressed, clear counts to 0
  Flag2 = 0
goto Loop

Count_it:
  if (in1 = 0) or (Flag1 = 1) then Loop ' If no longer pressed OR the
                                        ' flag is set, skip
  Counts = Counts +1                    ' Increment Counts
  Flag1 = 1                             ' Once Action executes, set Flag to 1
goto Loop

Clear_it:
  if(in2 = 1) or (Flag2 = 1) then Loop
  ' If no longer pressed,Or the flag is set, skip
  Counts = 0                            ' Clear counts to 0
  Flag2 = 1                             ' Prevents from clearing it again
  debug "Counter Cleared. Total Count = ", DEC Counts, 13
goto Loop
```

When PB1 is pressed, the program branches to the **`Count_it`** routine. Notice that the first line of this routine tests to see if the switch is open or **`Flag1`** is set. Neither is true upon the first pass through the program. Therefore, **`Counts`** is incremented, **`Flag1`** is set to 1 and program execution goes back to **`Loop`**. If PB1 still is being held down, **`Count_it`** is run again. This time, however, with **`Flag1`** set, the **`if-then`** statement sends the program back to **`Loop`** without incrementing **`Counts`** again. No matter how long the pushbutton is pressed, it will only register one "count" upon each closure. Although you are only incrementing the **`Count`** variable in this program, it could be part of a routine called for in an industrial application. Figure 2.9 is a screen shot that is representative of what you may see when running the program.

Figure 2.9: Running Program 2.4 - Edge Trigger Counting



Programming Challenge!

Use the indicating LEDs on output Pins 4 and 5, along with the two pushbuttons, to simulate a parking lot application. Assume your parking lot can hold 24 cars. Pushbutton PB1 will be counting cars as they enter the lot. Pushbutton PB2 will count cars as they leave. Write a program that will keep track of the total cars in the parking lot by counting "up" with PB1 and "down" with PB2. Have the green LED on as long as there is a vacancy in the lot. Turn the red LED on when the lot is full. Continually display how many parking spaces are available in the User Status window (!USRS). Plot continually the number of cars in the parking lot.

Additional StampPlot Lite Challenge

Keep a file of the number of times your parking lot went from "Vacancy" to "Full" (see Appendix A and the StampPlot Lite help file for information on using the Save Data to File option).

BUTTON Command: PBASIC's Debouncing Routine

Debouncing switches is a very common programming task. Parallax built into the PBASIC2 instruction set a command specifically designed to deal with digital input signal detection. The command is called **`button`**. The syntax for the command is shown below.

PBASIC Command Quick Reference: BUTTON

BUTTON pin, downstate,delay,rate,bytevariable,targetstate, address

- *Pin:* (0-15) The pin number of the input.
- *Downstate:* (0 or 1) Specifiying which logical state occurs when the switch is activated.
- *Delay:* (0-255) Establishes a settling period for the switch. Note: 0 and 255 are special cases. If delay is 0, Button performs no debounce or auto-repeat. If delay is 255, Button performs debounce but no auto-repeat.
- *Rate:* (0-255) Specifies the number of cycles between autorepeats.
- *Bytevariable:* The name of a byte variable needed as a workspace register for the BUTTON instruction.
- *Targetstate:* The state of the pin on which to have a branch occur.
- *Address:* The label to branch to when the conditions are met.

To try it with our counting routine, load and run program Program 2.5.

```
' Program 2.5: Button Exercise with StampPlot Interface
' Use Button to count and display the number of closures of PB1.
' Reset total count with a closure of PB2.

Pause 500
debug "!TITL Counting Challenge",13     ' Titles the StampPlot screen
debug "!TMAX 50",13                     ' Sets the plot time (seconds)
debug "!PNTS 300",13              ' Sets the number of data points
debug "!AMAX 20",13               ' Sets vertical axis (counts)
debug "!MAXR",13

Wkspace1  var  byte               ' Workspace for the BUTTON command for PB1
Wkspace1 = 0                      ' Must clear workspace before using BUTTON
Wkspace2  var  byte               ' Workspace for the BUTTON command for PB2
Wkspace2 = 0                      ' Must clear workspace before using BUTTON

Counts var word                   ' Word variable to hold count
Counts = 0
```

```
Loop:
  pause 50
  button 1,1,255,0,Wkspace1,1,Count_it
                                ' Debounced edge trigger detection of PB1
  button 2,0,255,0,Wkspace2,1,Clear_it
                                ' Debounced edge trigger detection of PB2

  debug "!USRS Total Count = ", DEC Counts,13
                                ' Display total counts in Status box
  debug dec Counts, 13          ' Show counts on analog trace
  debug ibin in1, bin in2,13    ' Plot the digital status.
goto Loop

Count_it:
  Counts = Counts +1                  ' Increment Counts
goto Loop

Clear_it:
  Counts = 0                    ' Clear counts to 0
  debug "Counter Cleared. Total Count = ", DEC Counts, 13
                                ' Display in Text Box
goto Loop
```

Review the documentation concerning the **`button`** command in the BASIC Stamp Programming Manual Version 1.9. This is a very handy command for industrial applications. Experiment by changing the delay time from 50 to 100 and to 200. See if you can press the switch more than one time but only get one **`Action`** to take place. What would be the risk of allowing for too much settling time in "high speed" counting applications? Save this program; it will be modified only slightly for use with the next programming challenge.

Electronic Digital Input Sources

It is very common for digital inputs to come from the outputs of other electronic circuits. These inputs may be from a variety of electronic sources, including inductive or capacitive proximity switches, optical switches, sensor signal-conditioning circuits, logic gates, and outputs from other microcontrollers, microprocessors, or programmable logic control systems.

There are several things to consider when interfacing these sources to the BASIC Stamp. Primarily, "Are they electrically compatible?"

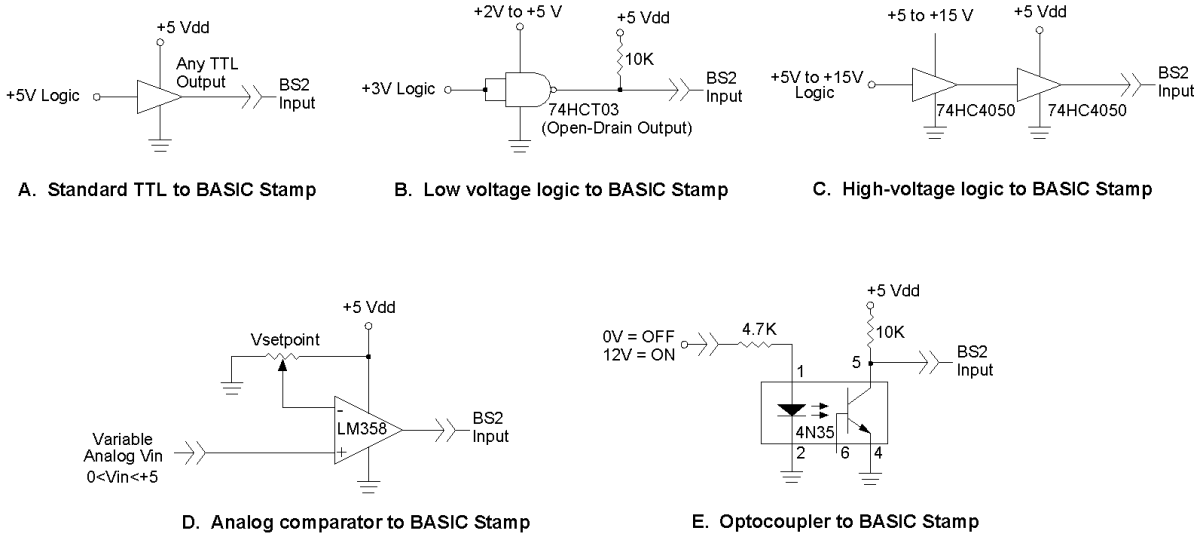
1. Is the source's output signal voltage within BASIC Stamp input limits?
2. Is the ground reference of the circuit the same as that of the BASIC Stamp?
3. Is protection of either circuit from possible electrical failure of the other a concern such that isolation may be necessary?

Once a compatible signal is established, the next question becomes, "Is the program compatible to respond to the input?"

1. Is digital bounce an issue?
2. How fast is the data? What is its frequency? What is the minimum pulse time?
3. Is action to be taken based on the data's steady-state level or on its leading or trailing edges?

Figure 2.10 shows a variety of electrical interfacing possibilities you may face.

Figure 2.10: Input Interfacing of Electronics to the BASIC Stamp



A. Standard TTL to BASIC Stamp

B. Low voltage logic to BASIC Stamp

C. High-voltage logic to BASIC Stamp

D. Analog comparator to BASIC Stamp

E. Optocoupler to BASIC Stamp

(a) TTL and CMOS logic inputs powered from a +5-volt supply can be applied directly to the BASIC Stamp's input pins. If the two systems are supplied from the same 5 volts, great. If not, at least the grounds must be common (connected together).

(b) Low-voltage (+3 V) devices can be interfaced using a 74HCT03 or similar open-drain gate with a pull-up resistor to the BASIC Stamp's +5-volt supply. Supply the chip with the low-voltage supply and make the grounds common.

(c) Higher-voltage digital signals can be interfaced using a 74HC4050 buffer or 74HC4049 inverter powered at +5 volts. These devices can safely handle inputs up to 15 volts. Again, the grounds must be common.

(d) A referenced comparator op-amp configuration can establish a High/Low output based on the analog input being above or below the setpoint voltage. The LM35810 is an op-amp whose output will go nearly rail to rail on a single-ended, +5-volt supply. It will be used in the upcoming application.

(e) An opto-coupler may be used to interface different voltage levels to the BASIC Stamp. The LED's resistor holds current to a safe level while allowing enough light to saturate the phototransistor. The input circuit can be totally isolated from the phototransistor's BASIC Stamp power supply. This isolation provides effective protection of each circuit in case of an electrical failure of the other.

Exercise #4: An Electronic Switch

Electronic switches that provide "non-contact" detection are very popular in industrial applications. No physical contact for actuation means no moving parts and no electrical contacts to wear out. The pushbutton switch used earlier should be good for several thousand presses. However, its return spring eventually will fatigue, or its contacts will arc, oxidize, or wear to the point of being unreliable.

Industrial electronic switches operate on one of three principles.

- Inductive proximity switches sense a change in an oscillator's performance when metal objects are brought near it. Most often the metal objects absorb energy via eddy currents from the oscillator causing it to stop.
- Capacitive proximity switches sense an increase in capacitance when any type of material is brought near them. When the increase becomes enough, it causes the switch's internal oscillator to start oscillating. Circuitry is then triggered and the output state is switched.
- Optical switches detect the presence or absence of a narrow light beam, often in the infrared range. In retroreflective optical switches, the light beam may be reflected by a moving object into the switch's optical sensor. Through-beam optical switches are set up such that the object blocks the light beam by going between the light source and the receiver.

The output of the an electronic switch is a bi-state signal. It's final stage may be any one of the types seen in Figure 2.10. As a technician and application developer, you must consider the nature of this signal circuit and condition it for the digital input of the microcontroller. The manufacturer's datasheet will give you information on the operating voltage for the switch and typical load connections.

Although you can think of the BASIC Stamp's digital input pin as the load, the electronic switch may require a reference resistor as used earlier in Figure #2.2. Most likely, the output of the proximity switch will be very near 0 volts in one state and near its supply voltage in the other state. It is always a good idea to test the switch's output states with a voltmeter before applying them to the unprotected input of the microcontroller. If the output voltages are not within the compatible limits of the BASIC Stamp, you will need to use one of the circuits in Figure 2.10 as an appropriate interface.

The following exercise focuses on the design and application of an optical switch. We will use this switch to detect and count objects. Then the switch will be used as a tachometer to determine RPM.

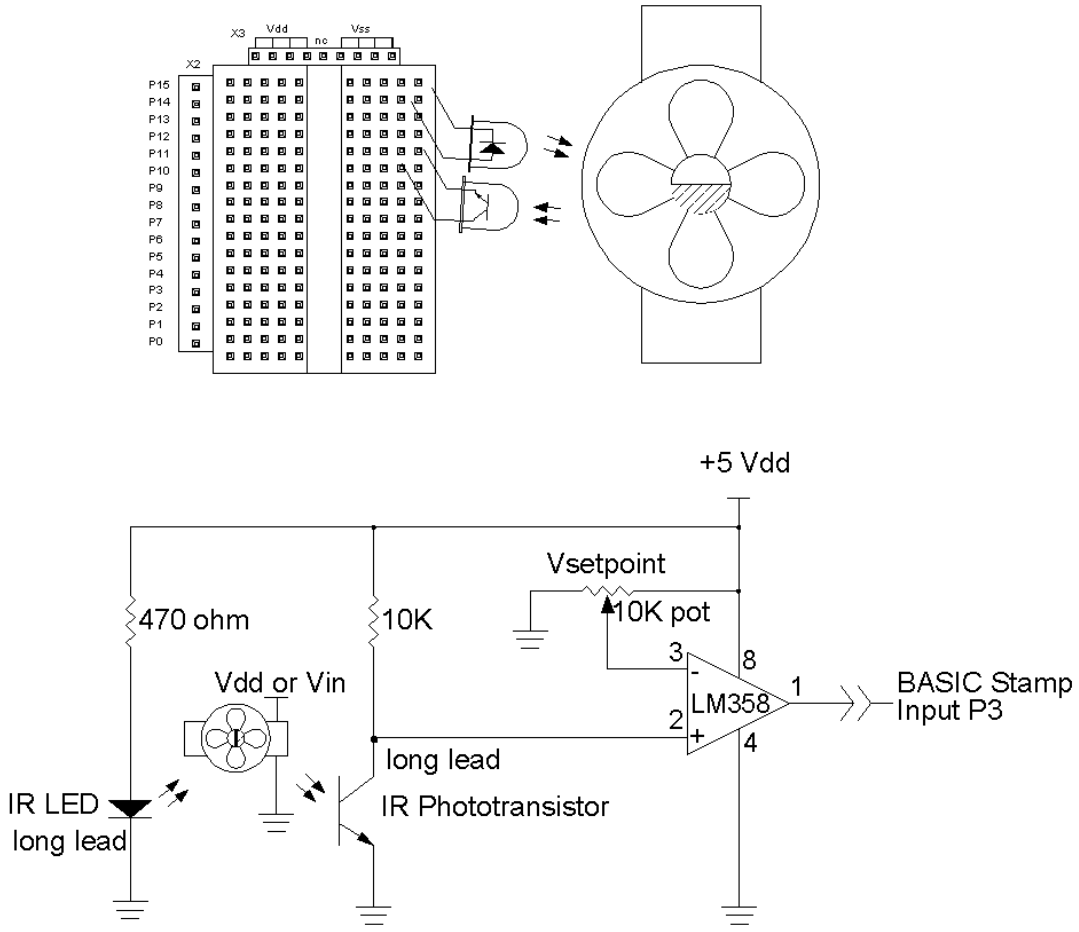
In Figure 2.11, the infrared light-emitting diode (LED) and the infrared phototransistor form a matched emitter/detector pair. Light emitted by the LED will result in phototransistor collector current. An increase in collector current drives the phototransistor toward saturation (ground). If the light is prevented from striking the phototransistor, it goes toward cutoff and the collector voltage increases positively. These conditions of light and no-light will most likely not provide a legal TTL signal at the collector of the transistor. Applying this signal to the input of a referenced comparator will allow us to establish a setpoint somewhere between the two conditions. The output of the comparator will be a compatible TTL logic signal. It's level is dependant on which side of the setpoint the phototransistor's output is on. The LM358 op-amp is a good choice for this application. It can operate on a +5-volt single supply and its output saturation voltages are almost equal to the supply potentials of +5 and ground.

Carefully construct the circuit in Figure 2.11 on the Board of Education breadboard. Mounting the devices near one end as pictured in the diagram allows for additional circuits in upcoming exercises. Make a 90^{o} bend in the LED and phototransistor leads so the devices lie parallel to the the benchtop. The phototransistor and infrared LED should be placed next to each other, pointing off the edge of the breadboard.

The LED in Figure 2.11 is emitting a continuous beam of infrared light. With the LED and phototransistor side-by-side, there is little or no light coming into the phototransistor because there is nothing reflective in front of it. If an object is brought toward the pair, some of the LED light will bounce back into the phototransistor. When light strikes the phototransistor, the collector current will flow and the collector voltage will drop. In this setup, the scattered reflection of light off an object as it passes in front of the pair will be sensed by the phototransistor. The amount of reflected light into the sensor depends on the optical reflectivity of the target object and the geometry of the light beam. We will attempt to determine the presence of a flat-white object. With the emitter and detector mounted side-by-side, you will try for detection of the object at a distance of one inch.

You must make a couple of voltage measurements to calibrate the presence of the object. Begin by placing a voltmeter across the phototransistor's collector and emitter. Measure the voltage when there is no object in front of the sensor. Record this value in Table 2.3. Next, move a white piece of paper toward and away from the pair and notice the variation in voltage. As the paper is brought near the IR pair, the reflected light increases collector current and drives the transistor toward saturation –"low." Record the voltage reading with the white paper approximately one inch in front of the sensor in Table 2.3. The difference between these measurements may be quite small, like 0.5 V, but that will be enough to trigger the op-amp. This signal is applied to the inverting input of the LM358 comparator. The potentiometer provides the non-inverting input reference voltage. This reference should be a value between the "no reflection" and "full reflection" readings. Adjust the potentiometer to provide the proper reference voltage, which is halfway between the measurements.

Figure 2.11: Retro-reflective Switch Pictorial and Schematic



Testing the output of the LM358 should result in a signal compatible with the BASIC Stamp. The output should be low with no object and high when the white object is placed in front of the emitter/detector pair. Measure these two output voltages of the LM358 and record the values in Table 2.3. If the output signal is compatible, apply it to the BASIC Stamp's Pin 3. Detecting light reflected by an object is called retro-reflective detection.

Table 2.3: LM358 Values

Condition	Phototransitor Voltage	LM358 Output Voltage
No object – no reflection		
Object – full reflection		
Reference voltage setpoint		

This ability to yield a switching action based on light received lends itself to many industrial applications such as product counting, conveyor control, RPM sensing, and incremental encoding. The following exercise will demonstrate a counting operation. You will have to help, though, by using your imagination.

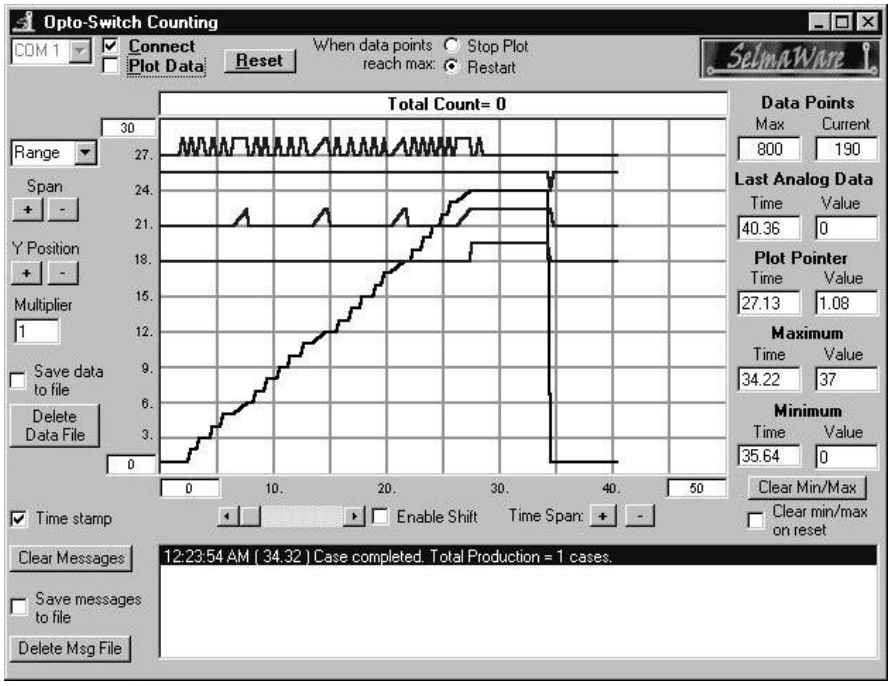
Let's assume that bottles of milk are being transferred on a conveyor between the filling operation and the case packer. Cut a strip of white paper to represent a bottle of milk. Passing it in front of our switch represents a bottle going by on the conveyor. Only a slight modification of the previous program is necessary to test our new switch. If you have Program 2.5 loaded, simply modify the first **`button`** instruction by changing the input identifier from Pin 1 to 3. The modified line would look like this:

```
' Program 2.6 (modification to Program 2.5
' for the retroreflective switch input)

button 3,1,255,0,Wkspace1,1,Count_it
' Debounced edge trigger detection of optical switch
```

Programming Challenge: Milk Botttle Case Packer

Refer back to Experiment #1 and consider the conveyor diverter scenario in Figure 1.2. We will assume that the controller is counting white milk bottles and controlling a diverter gate and a case packer drop chute. Use the green and red indicating LEDs attached to the StampPlot Lite output Pins 4 and 5 to represent these outputs. Modify your program to count bottles of milk on the conveyor. Let the diverter gate be used to direct the bottles to a case-packing machine. Change the status of the diverter each time six bottles have been counted. Upon 24 bottles being counted (the 4th six-pack), turn the "case packer" LED on for two seconds. Use the StampPlot Lite interface to monitor the operation. Report to the User Status window the present count of bottles in the case. Monitor the bottle count and the digital output status using the analog and digital graphs. Clear it after each case. In the lower Text Box, time-stamp each time a case is filled and report how many total cases have been processed. The interface should appear similar to the example screen shot in Figure 2.12.

Figure 2.12: Example Screen Shot of Milk Bottle Challenge



Exercise #5: Tachometer Input

Monitoring and controlling shaft speed is important in many industrial applications. A tachometer measures the number of shaft rotations in a unit of time. The measure is usually expressed in revolutions per minute (RPM).

A retroreflective switch can open and close fast enough to count white and black marks printed on a motor's shaft. Counting the number of closures in a known length of time provides enough information to calculate RPM. Figure 2.13 represents five possible encoder wheels that could be attached to the end of a motor shaft. If the optical switch is aimed at the rotating disk, it will pulse on-off with the alternating segments as they pass. The number of white (or black) segments represent the number of switch cycles per revolution of the shaft. The first encoder wheel has one white segment and one black segment. During each revolution, the white segment would be in front of our switch half the time, resulting in a logic high for half the rotation.

During the half rotation the black segment is in front of the disk, it absorbs the infrared light and with no reflected light, the switch will be low. One cycle of on-off occurs each revolution. The PBASIC2 instruction set provides a very useful command called **count** that can be used to count the number of transitions at a digital input occuring over a duration of time. Its syntax is shown below.

PBASIC Command Quick Reference: COUNT

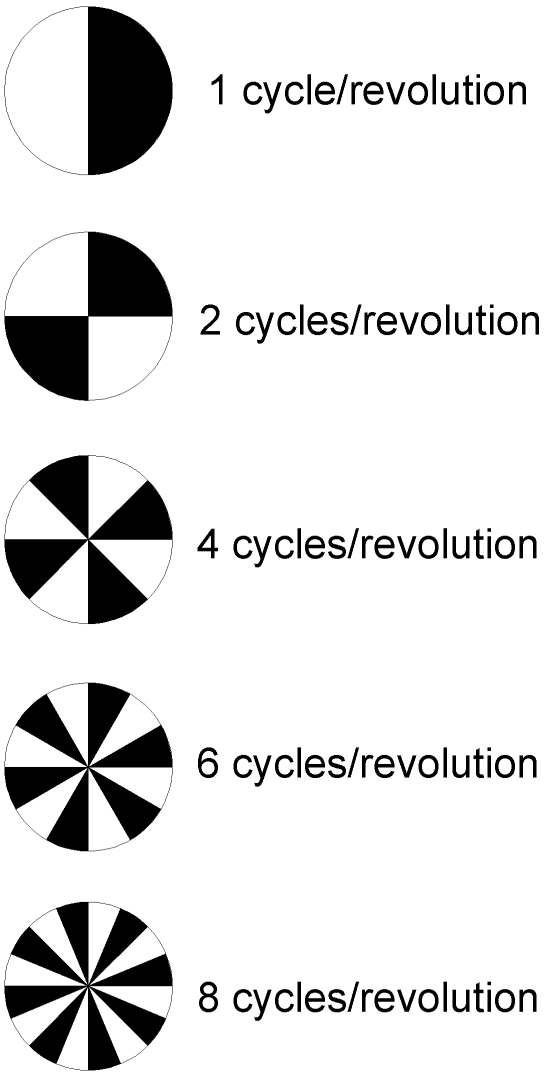
COUNT pin, period, variable

- *Pin: (0-15) Input pin identifier.*
- *Period: (0-65535) Specifies the time in milliseconds during which to count.*
- *Variable: A variable in which the count will be stored.*

The following exercise uses the count instruction, the optical switch, and the shaft encoder wheels to capture speed data.

Begin by cutting out the first encoder wheel. Use a small piece of cellophane tape to hold the encoder to the shaft hub of the fan motor (a full-size encoder wheel set may be pulled from Appendix B of this text). Wire a 10K-ohm trimpot in series with your fan so its speed can be varied. The fan is rated at 12 V. Its speed changes with varying voltages from 12 V down to approximately 3.5 V. This is the dropout voltage of the brushless motor control circuitry. The fan should be located so the encoder wheel is pointed at the emitter/detector pair. Pin 20 of connector X1 provides access to the 12-volt unregulated supply.

Figure 2.13: Retro-reflective Encoder Wheels
(cutouts are available in Appendix B)



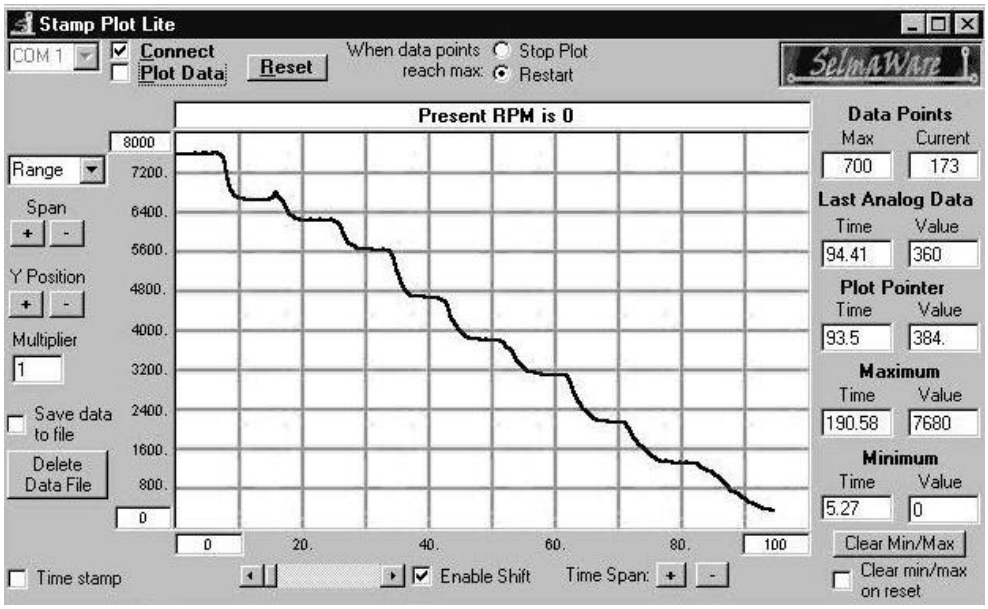
The first encoder wheel has one white and one black segment on it. As it rotates, the opto-switch should cycle on-off once for each revolution. Enter the Tachometer Test Program 2.7 below.

```
' Program 2.7 Tachometer Test - with the StampPlot Interface

' Initialize plotting interface parameters.
' (Can also be set or changed on the interface)
debug "!AMAX 8000",13            ' Full Scale RPM
debug "!AMIN 0",13               ' Minimum scaled RPM
debug "!TMAX 100",13             ' Maximum time axis
debug "!TMIN 0",13               ' Minimum time axis
debug "!AMUL 1",13               ' Analog scale multiplier
debug "!PNTS 600",13             ' Plot 600 data points
debug "!PLOT ON",13              ' Turn plotter on
debug "!RSET",13                 ' Reset screen

Counts var word                  ' Variable for results of count
RPM var word                     ' Variable for calculated RPM
Counts = 0                       ' Clear Counts

Loop:
  count 3,1000, Counts           ' Count cycles on pin 3 for 1 second
  RPM = Counts * 60              ' Scale to RPM
                                 ' Send out RPM value to plotter and status bar
  debug dec rpm,13
  debug "!USRS Present RPM is ", DEC RPM,13
goto Loop
```

As the fan spins, the cycling of the photo switch will be counted for 1000 milliseconds (one second). With the duration of the **`Counts`** routine being one second and one cycle occurring with each rotation, we get the cycles per second of fan rotation. Most often, the speed of a rotating shaft is described in terms of revolutions per minute (RPM). Multiplying the revolutions per second times 60 converts cycles per second to RPM.

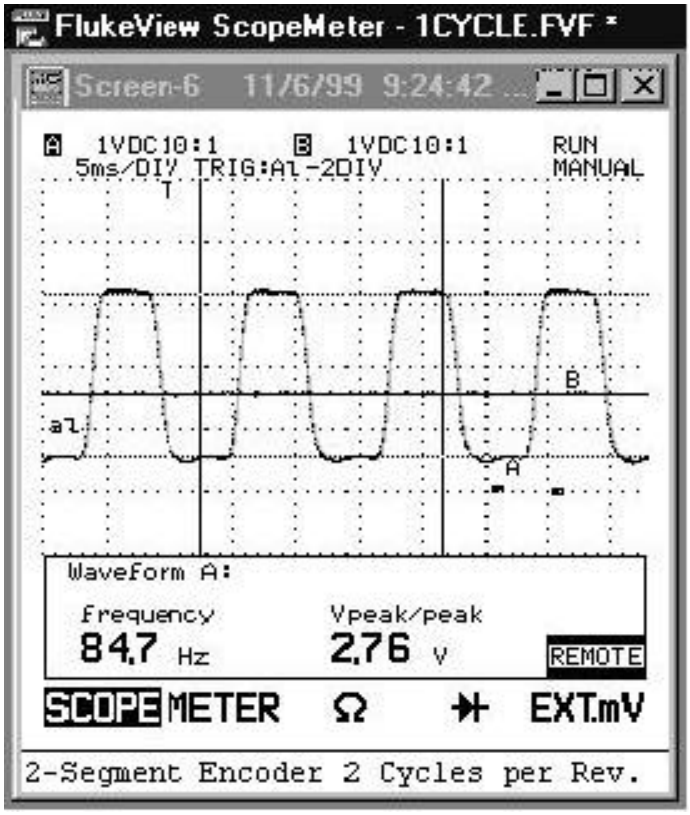
Run your program. The debug window will first appear with the serial information for configuration and display of the StampPlot Lite interface. Close the BASIC Stamp debug window and open StampPlot Lite. Check "Connect and Plot Data" and click on "Restart." Press the Board of Education reset button and your interface should start plotting. Figure 2.14 shows a representative screen shot of the interface plotting RPM at various motor voltages.

Figure 2.14: RPM of the Brushless DC Fan at Varying Voltages



The spinning encoder wheel may result in a slightly different phototransistor output for "light" and "no-light" conditions. If your system is not reporting correctly, change the setpoint by adjusting the potentiometer to the new average value. If you have access to an oscilloscope, measure the peak-to-peak output of the phototransistor and your potentiometer setpoint being applied to the comparator. Placing the setpoint midway between the peak-to-peak DC voltage levels would allow for optimal performance. Notice the frequency and wave shape of the signal. An example of the oscilloscope reading is pictured in Figure 2.15. The 84.7 Hz equated to a debug readout of "Counts = 84 RPM = 5040." The 84.7 Hz measured by the oscilloscope reflects an actual RPM of 84.7 x 60 = 5,082. Only 84 complete cycles fell within the one-second capture time of our routine.

Figure 2.15: Two-Segment Encoder Oscilloscope Trace



Record your tachometer readout when maximum voltage is applied to the motor. You can use the Board of Education's Vin (unregulated 9 V) for high speed, or the Vdd (regulated 5 V) for different speeds.

Counts = _____ RPM = _____

When testing your tachometer, notice the effects of slowing the motor with slight pressure from your finger. The counts will decrease by factors of one. In the Figure 2.13 example, it would decrease from 83 to 82 to 81, etc., and the resulting RPM readings drop by a factor of 60 (4980 to 4920 to 4860, etc.).

Because we are counting for one second and we get one cycle per revolution, the program can resolve RPM only to within an accuracy of 60. To get a more accurate assessment of RPM, you have a couple of choices: increase the time you count cycles, or increase the cycles per revolution.

Let's try the first choice. Increase the count time in Program 2.7 from 1000 milliseconds to 2000 milliseconds. By doing so, you are now reading during a two-second window and RPM is equal to 30 times this value. {(Counts/2 seconds) x 30 = RPM} and resolution is now to within 30 RPM. Change the scaling value for RPM from 60 to 30 in the **`RPM =`** line of the program.

Increasing the count duration time increases the accuracy of the RPM reading. Refer to Table 2.4.

Table 2.4: Given Encoder Frequency of 84.7 Hz
From the 1 cycle/second Encoder is an RPM of 5082

Duration	Counts	Scaler	RPM
1000 mS	84	60	5040
2000 mS	169	30	5070
3000 mS	254	20	5080
60000 mS	5082	1	5082

As you can see, to gain a resolution of one RPM, our count routine had to be one full minute (60,000) in duration. Unless you are very patient, this is unacceptable! In terms of programming, the BASIC Stamp is tied up with the **`count`** routine for the total duration. During this time, the rest of the program is not being serviced. For this reason, long duration also is not good.

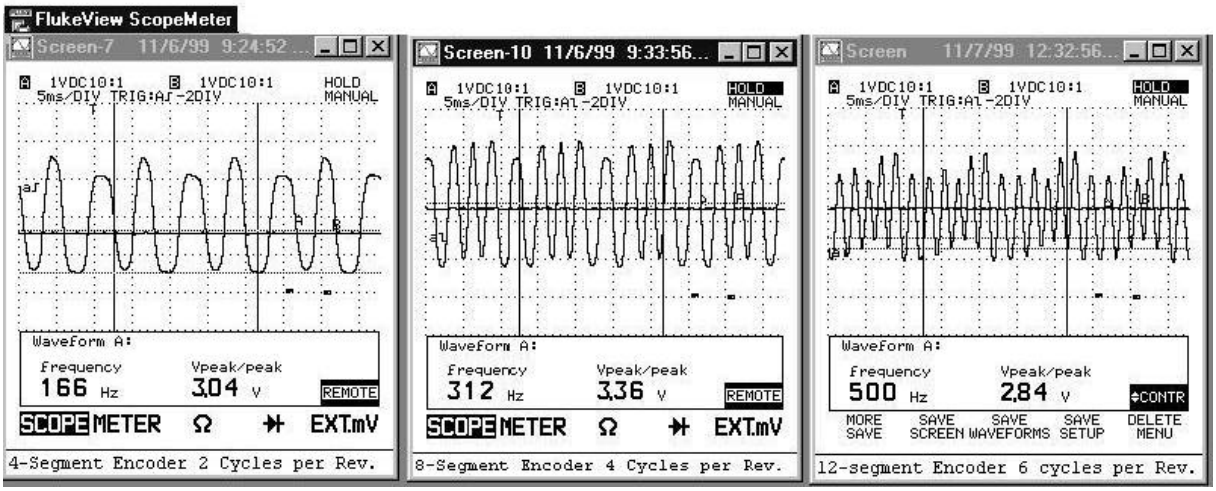
Another method of improving resolution is to increase the number of cycles per revolution. Cut out the second encoder wheel and tape it to your fan motor hub. This wheel has two white segments and will produce two count cycles per revolution. During a one-second-count duration, this encoder will produce twice as many pulses as the first encoder. The RPM calculation line of the code would be **`RPM = Counts x 60 / 2`** for this encoder, or **`RPM = Counts x 30`**. Try it!

The third encoder wheel yields even more resolution by with four cycles per revolution. Tape this encoder to your motor's hub and change the program's RPM line to **`RPM = Counts * 15`**. You may have to vary the setpoint potentiometer as you switch from one encoder wheel to another.

If you use the six-cycle encoder, what value would you use to scale the Counts to RPM? Fill in your answer in Table 2.5.

Figure 2.16 includes oscilloscope traces recorded from using the two-cycle, four-cycle, and six-cycle encoder wheels on a shaft rotating at 4,980 RPM. It is the focal properties of the emitter/detector pair that will limit the maximum number of segments on the encoder wheel. You may find it difficult to use the six-cycle encoder wheels without devising some sort of shielding and/or focusing of the light beam.

Figure 2.16: Two-cycle, Four-cycle, and Six-cycle Encoder Wheel Oscilloscope Traces



The accuracy required of a tachometer system is dependent on the application. Commercial shaft encoders are available with resolutions greater than 500 counts per revolution. Fill in the appropriate values in Table 2.5 for an encoder with a resolution of 360 counts per revolution.

Table 2.5: Given a Shaft Speed of 4,980 RPM

Cycles per Revolution	Counts	Scaler	RPM
1	83	60	4,980
2	166	30	4,980
4	312	20	4,980
6	498	_____	4,980
360	_____	_____	_____

Challenge! Analyze your Motor's Characteristics

It would be interesting to determine the speed voltage characteristics of your motor. Beginning at full voltage, adjust the motor's voltage to several different levels throughout its operating range. Allow the motor to stabilize at each voltage level and record those levels. Be quick or increase the time span of StampPlot Lite. Once you have gone through the range from full voltage to stall voltage, stop the plot. Use the mouse cursor to read the stable RPM at each step and record the RPM alongside its corresponding voltage. Plot your data and summarize the speed/voltage characteristics of your motor. Figure 2.14 is a screen shot resulting from performing this challenge. The motor we used had fair linearity for voltages from 6V to 12V. At low voltages, its linearity was poor, and it stalled at 3.4 V.

Questions and Challenge

Questions

1. An industrial device whose output is either one of two possible states is termed _____.

2. What is the "ideal" resistance of a mechanical switch in the open state? In the closed state?

   Open-state resistance = _____ and, Closed-state resistance = _____

3. Explain the purpose of placing a resistance in series with a switch for conditioning a digital input signal.

4. A normally-open pushbutton switch configured in an "active low" state will be read as a logic _____ when not being pressed.

5. What is the absolute maximum input voltage to the BASIC Stamp?

6. For some CMOS devices, an input of 1.3 volts is in the _____ area of operation.

7. Low-voltage logic devices operate on _____ volts DC.

8. What type of proximity switch activates only on metal objects?

9. When light strikes the base of a phototransistor, the collector current will _____ and collector to emitter voltage will _____.

10. A car's six-cylinder engine RPM can be determined by counting the pulses delivered to the ignition coil. Six pulses are required for one revolution. If 20 pulses occur in one second, what is the RPM of the engine?

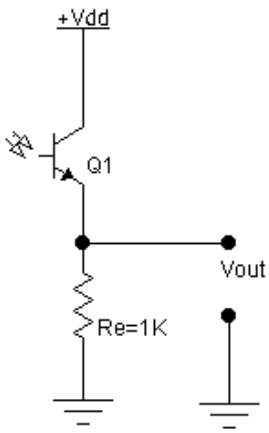
Design it!

1. Draw a diagram of a normally-open pushbutton switch and its "pull-up" resistor. The diagram should be drawn so pressing the switch results in a logic "low" output.

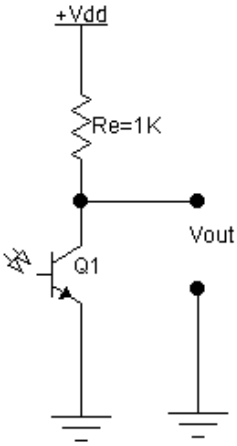
2. Draw a diagram of a normally-closed pushbutton switch and a 10K-ohm series resistor. The diagram should be drawn so pressing the switch results in a logic-low output.

Analyze it!

1. Consider the two phototransistor circuits below. Which one has an increasing output voltage with increases in light level? Why? What is the output voltage of Circuit B if the light level saturates transistor Q1?

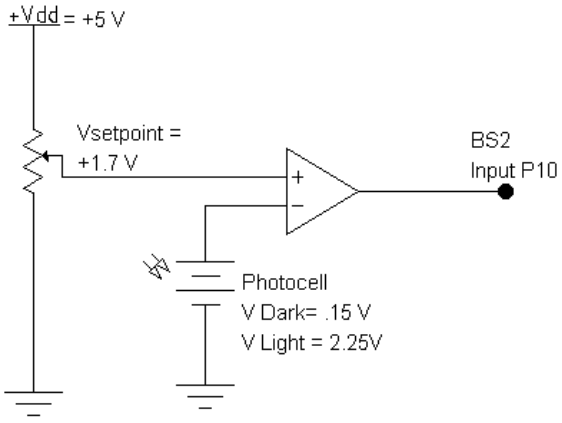


Circuit A



Circuit B

2. The comparator circuit below is used to determine when to turn on and off a dusk-to-dawn security lamp. What would be the output status of the comparator during "light" conditions? Would it be better to program for detecting the voltage level or the edge triggering of this circuit? Why?



Night time security light.

Program it!

1. Pretend that your retro-reflective tachometer is providing the input to an anti-lock braking system on an automobile. In conjunction with this input, use a pushbutton to model the brake pedal switch. An active high LED will represent the braking action. Write a program that will detect the pressing of the brake pedal and slow the motor, also turning on the LED as long as speed is above zero. When shaft speed drops to zero, turn off the LED. Use a potentiometer to set initial motor speed. Configure the two pushbutton switches as active-high inputs. Wire one LED as an active-high output.

2. Write programs to duplicate the operation of an OR, AND, and XOR gate.

OR Gate		
PB1	PB2	LED
0	0	0
1	0	1
0	1	1
1	1	1

AND Gate		
PB1	PB2	LED
0	0	0
1	0	0
0	1	0
1	1	1

XOR Gate		
PB1	PB2	LED
0	0	0
1	0	1
0	1	1
1	1	0

Field Activity

How many digital (bi-state) field devices can you identify in a new car? List as many as you can. Make a note as to whether you suspect that the field device directly controls load current , drives some sort of relay, or if you think its status is being monitored by a microcontroller.